Voici un tutoriel pour créer une application CRUD en ReactJS qui consomme l'API fournie (https://caa615be5646d221bd51.free.beeceptor.com/api/utilisateurs/) et utilise Bootstrap pour le design :

Prérequis :

1. Avoir Node.js et npm installés sur votre machine.
2. Connaître les bases de ReactJS et avoir une installation de base de React (si ce n'est pas déjà fait, créez une application React en utilisant `npx create-react-app crud-app`).

L'objectif est de créer une application simple de gestion des utilisateurs où vous pouvez ajouter, modifier et supprimer des utilisateurs.

Étape 1 : Créer un nouveau projet React

1. **Créer le projet React** : Utilisez la commande suivante pour créer un nouveau projet React :

```
npx create-react-app gestion-utilisateurs
```

2. **Naviguer vers le répertoire du projet** :

```
cd gestion-utilisateurs
```

3. **Installer Bootstrap** pour styliser l'application :

```
npm install bootstrap
```

   Ensuite, ajoutez `import 'bootstrap/dist/css/bootstrap.min.css';` dans votre fichier `src/index.js` ou `src/App.js` pour charger les styles Bootstrap.

Étape 2 : Structure du projet

Voici la structure du projet que nous allons créer :

```
src/
├── components/
│   ├── UserForm.js
│   ├── UserList.js
├── App.js
└── api.js
```

Étape 3 : Explication des fichiers

1. **Fichier `src/api.js`** :

   Ce fichier contient l'URL de l'API. Nous allons l'utiliser pour gérer les appels API.

```
// src/api.js
```

```
export const API_URL =
'https://ca1be343299cbe7b2886.free.beeceptor.com/api/utilisateurs/';
```

Explication :

- Ce fichier contient simplement une variable `API_URL` qui stocke l'URL de notre API. Cette URL sera utilisée pour faire des requêtes HTTP dans les autres composants.

---

2. **Fichier `src/components/UserList.js` :**

   Ce fichier est responsable de l'affichage de la liste des utilisateurs. Il utilise `axios` pour récupérer les données depuis l'API et afficher les utilisateurs dans un tableau.

```
// src/components/UserList.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { API_URL } from '../api';  // Importer l'URL ici

const UserList = ({ setUserToEdit }) => {
  const [users, setUsers] = useState([]);

  const refreshUsers = () => {
    axios.get(API_URL)
      .then(response => {
        setUsers(response.data);
      })
      .catch(error => console.error('Erreur lors de la récupération des
utilisateurs:', error));
  };

  useEffect(() => {
    refreshUsers();
  }, []);

  const handleDelete = (id) => {
    axios.delete(`${API_URL}${id}`)
      .then(() => {
        setUsers(users.filter(user => user.id !== id));
      })
      .catch(error => console.error('Erreur lors de la suppression de
l\'utilisateur:', error));
  };

  return (
    <div >
      <h2>Liste des utilisateurs</h2>
      <button className="btn btn-info mb-3"
onClick={refreshUsers}>Actualiser</button>
      {Array.isArray(users) ? (
        <table className="table table-striped">
          <thead>
            <tr>
              <th>#</th>
              <th>Nom</th>
              <th>Email</th>
```

```
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {users.map(user => (
              <tr key={user.id}>
                <td>{user.id}</td>
                <td>{user.nom}</td>
                <td>{user.email}</td>
                <td>
                  <button className="btn btn-primary" onClick={() =>
setUserToEdit(user)}>Modifier</button>
                  <button className="btn btn-danger" onClick={() =>
handleDelete(user.id)}>Supprimer</button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      ) : (
        <p>Les données ne sont pas disponibles sous forme de tableau.</p>
      )}
    </div>
  );
};

export default UserList;
```

Explication :

- **useState** : Ce hook est utilisé pour gérer l'état des utilisateurs (`users`).
- **useEffect** : Ce hook effectue la récupération des utilisateurs lorsque le composant est monté pour la première fois.
- **axios.get()** : Cette fonction permet de récupérer les utilisateurs de l'API.
- **handleDelete** : Cette fonction permet de supprimer un utilisateur en envoyant une requête `DELETE` à l'API.
- **setUserToEdit** : Cette fonction est passée en prop et permet de définir un utilisateur à modifier.

---

3. **Fichier `src/components/UserForm.js` :**

   Ce fichier est responsable du formulaire pour ajouter ou modifier un utilisateur.

```
// src/components/UserForm.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { API_URL } from '../api';  // Importer l'URL ici

const UserForm = ({ userToEdit, refreshUsers }) => {
  const [formData, setFormData] = useState({ nom: '', email: '' });

  useEffect(() => {
    if (userToEdit) {
      setFormData({ nom: userToEdit.nom, email: userToEdit.email });
    }
```

```
  }, [userToEdit]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    const method = userToEdit ? 'put' : 'post';
    const url = userToEdit
      ? `${API_URL}${userToEdit.id}`
      : API_URL;

    axios[method](url, formData)
      .then(() => {
        refreshUsers();
        setFormData({ nom: '', email: '' });
      })
      .catch(error => console.error('Erreur lors de l\'ajout ou
modification de l\'utilisateur:', error));
  };

  return (
    <div >
      <h2>{userToEdit ? 'Modifier l\'utilisateur' : 'Ajouter un
utilisateur'}</h2>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Nom</label>
          <input
            type="text"
            name="nom"
            className="form-control"
            value={formData.nom}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Email</label>
          <input
            type="email"
            name="email"
            className="form-control"
            value={formData.email}
            onChange={handleChange}
            required
          />
        </div>
        <button type="submit" className="btn btn-success mt-3">
          {userToEdit ? 'Modifier' : 'Ajouter'}
        </button>
      </form>
    </div>
  );
};

export default UserForm;
```

Explication :

- **useState** : Gère l'état du formulaire pour le nom et l'email.
- **useEffect** : Ce hook est utilisé pour remplir le formulaire si un utilisateur est sélectionné pour modification.
- **handleChange** : Gère les changements dans les champs de formulaire.
- **handleSubmit** : Gère l'envoi du formulaire. Si un utilisateur est modifié, une requête `PUT` est envoyée, sinon une requête `POST` est envoyée pour ajouter un utilisateur.

---

4. **Fichier `src/App.js` :**

   Ce fichier est le composant principal de l'application. Il contient les autres composants `UserList` et `UserForm`.

```
// src/App.js
import React, { useState } from 'react';
import UserList from './components/UserList';
import UserForm from './components/UserForm';
import 'bootstrap/dist/css/bootstrap.min.css';

const App = () => {
  const [userToEdit, setUserToEdit] = useState(null);

  const refreshUsers = () => {
    setUserToEdit(null); // Réinitialise le formulaire après l'ajout ou la
modification
  };

  return (
    <div className="container">
      <h1>Gestion des utilisateurs</h1>
      <div className="row">
        <div className="col-md-4">
          <UserForm userToEdit={userToEdit} refreshUsers={refreshUsers} />
        </div>
        <div className="col-md-8">
          <UserList setUserToEdit={setUserToEdit} />
        </div>
      </div>
    </div>
  );
};

export default App;
```

Explication :

- Le composant `App` contient deux composants enfants : `UserForm` pour le formulaire d'ajout et de modification et `UserList` pour afficher la liste des utilisateurs.
- **useState** : Gère l'état de l'utilisateur à modifier (si aucun utilisateur n'est sélectionné, c'est `null`).
- **refreshUsers** : Cette fonction est utilisée pour réinitialiser le formulaire après qu'un utilisateur a été ajouté ou modifié.

Étape 4 : Démarrer l'application

1. **Lancer l'application** :

```
npm start
```

Cette commande démarre le serveur de développement et ouvre votre application dans le navigateur. Vous devriez voir une interface où vous pouvez ajouter, modifier et supprimer des utilisateurs.